ADAPTIVE ACTION PROMPTING — A COMPLEMENTARY AID TO SUPPORT TASK-ORIENTED INTERACTION IN EXPLORATIVE USER INTERFACES

Thomas Kühme

Siemens Corporate Research and Development
Otto-Hahn-Ring 6, 8000 Munich 8, Germany

Email: kuehme@zfe.siemens.de

ABSTRACT

Adaptive action prompting supports users by suggesting how to continue with the interaction. A small number of continuously updated action prompts are offered in addition to the regular menu interface of an application. Users can use these prompts either occasionally in trouble situations or for a sequence of system-guided action selections.

The suggestions are based on models of the application, context, and user. According prompting strategies are automatically evaluated and can optionally be controlled by the user. Multimodal user interfaces provide further perspectives for adaptive prompting.

Keywords: Adaptive User Interfaces, Intelligent User Interfaces, Intelligent Agent, Application Model, User Model, Multimodal Interfaces.

INTRODUCTION

In contemporary user interfaces extensive prompting is provided by means of icons, menus, buttons, input fields, and many other, often application specific controls. Due to the inherently large number of prompts, not all them can be made immediately available at the same time. Thus, users have to accomplish the task of searching for the appropriate prompt in a structured environment of hierarchic windows, lists, menus, and dialog boxes. Attempts to avoid hierarchical structure, on the other hand, lead to confusingly large numbers of items presented in a single menu or dialog box.

Action selections from menus involve particular problems. Menu hierarchies typically include a menu bar with a couple of pull-down submenus with several items each some of which are further structured through cascaded menus. At the same time, interfaces often provide pop-up menus, usually one in each window pane and also sometimes cascaded. Navigating through these hierarchies in order to locate and select the desired action can be a time-consuming process, even if the user does know where exactly to find the item in question– what may or may not be the case.

But users not only have problems to access the actions they already know about. Prior to selecting an action, users have to consider if the application provides an appropriate action at all. Even more basically, users have to imagine what would be an appropriate action to proceed with in order to perform a certain task of the application domain (cf. [10]). While in direct manipulation interfaces all available functions are principally accessible at the same time, thus allowing for an explorative working style, these interfaces generally do not support users in dealing with such sequencing problems.

Adaptive action prompting, as proposed in this paper, provides the user with a small number of immediate prompts for the most appropriate actions in a given situation. The underlying prompting strategies are adaptive about the changing context of both interaction and application and about the evolving knowledge and preferences of an individual user. The prompts are offered as a complementary aid, neither replacing nor interfering with the existing menus. Thus, explorative working in a rich environment is always possible if desired but not necessary in situations with a clear focus in the task domain.

While this paper concentrates on the technical aspects of adaptive action prompting an earlier paper [7] describes the general idea of adaptive prompting as applicable to different concepts in contemporary user interfaces, including tool selection (see also [8]) and dialog boxes (see also [11]).

First, we provide a brief discussion of related work. Then, the adaptive action prompter is described in terms of its user interface and its functional characteristics. The underlying context model and the embedded prompting strategies are subsequently discussed in detail. Next, we show how the prompting strategies are partly controlled by the user and how they can be evaluated through built-in mechanisms. The discussion of limitations and risks of adaptive prompting is followed by an outlook on further implications of this work for multimodal interfaces. We conclude with the current status and future work.

RELATED WORK

Various attempts have been made to reduce the navigation effort in menu hierarchies. Shortcuts, for instance, allow for a fast random access to menu items through pressing certain keys in combination or sequence. However, shortcuts trade faster accessibility for again more artifacts users have to remember. Pointer setting strategies for pop-up menus provide for a faster access to a certain menu item, often the most recently or most frequently used.

Other approaches try to support users in selecting valid and appropriate items. Grey-shading of disabled menu-items falls into this category. Often, actions are prompted according to the previously selected objects, for instance, in object-specific menus (most typically pop-up menus of editors and browsers) or dynamically exchanged control panels (e.g., in computer-aided design systems).

In some environments, menus are configurable, i.e. users can determine which items are contained in a menu and in which order. Thus, users have the opportunity to adapt the menus to their particular needs and preferences. Microsoft Word [12] provides an adaptive menu which offers the most recently opened files. To reopen these files, users simply choose from the menu rather than going through the dialog with a file selection box.

Related research focuses on a reorganization of menus according to user-specific usage patterns. Balint [1], for instance reports on an adaptive dynamic menu system where positions of menu items are automatically adjusted after every interaction. Sukaviriya and Foley [16], in contrast, propose to explicitly suggest modifications to menu layouts to users. The common goal is an increased performance due to the fact that the most frequently used item is moved to the most convenient location to select. Both approaches differ from the work described in this paper in that they modify the otherwise static menus of an application and that they only consider the statistical history of interactions to infer an improved menu structure.

Cypher [3] introduced Eager, a Programming-By-Example system to automate iterative patterns in interaction. Eager is similar to adaptive prompting in that it anticipates which action the user is going to perform next. Whenever detecting a repetitive pattern in the dialog, it simultaneously shows the user how it would proceed by turning the corresponding menu items, buttons, etc. green. The user simply clicks on the Eager icon for task completion at any point in time. With its focus on repetitive tasks, a mechanism like that of Eager could be a supplemental part of adaptive prompting.

The user interface environment UIDE [18] supports users in action selection by providing context-sensitive help on why an action is currently disabled and how to enable this action. Corresponding help can be either textual or animated [17] and is generated from a model of the application and interaction. The work described in this paper is closely related to UIDE and is based on its models and built-in provision for collecting individual task usage information [16]. A comparison between context-sensitive help and adaptive prompting is provided in a later section.

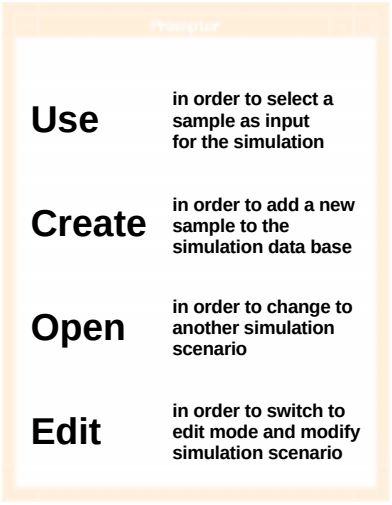

Figure 1. The Adaptive Action Prompter

THE ADAPTIVE ACTION PROMPTER

A prompter in this context is an intelligent agent who helps the user by suggesting how to continue with the interaction. Unlike prompting an actor in a play, where a singular prompt is the only appropriate one in a given situation, mostly there are several alternative ways to continue with in a human-computer dialog. Accordingly, the action prompter provides a choice of prompts rather than only one. Like the real prompter, the action prompter keeps track of the context and simultaneously exchanges and adapts its suggestions.

The prompts are presented as a menu in a small, permanently visible window (figure 1). This is in addition to the regular user interface the application provides. No changes – neither static nor dynamic– are made to this interface. Users are free to select actions from either the prompter menu or the menus of the original interface.

Whenever the user does not know how to proceed with the interaction the prompter can be consulted and used for an action selection. In situations with a clear focus, the action prompter allows for temporarily switching to an ATM (automatic teller machine) kind of interaction. While working towards a certain goal or a couple of related goals users can make a series of subsequent selections from the prompter. Returning to a more explorative dialog in the regular interface is straightforward and always possible. Thus, the prompter supports a smooth change between arbitrary dialogs and a system-guided interaction.

User Interface

Obviously, the user interface to the action prompter needs to be as simple and easy to understand as possible. Otherwise the prompter would impose new problems instead of supporting users. However, the success of adaptive prompting supposedly depends on how well it is adjusted to the individual user. It is unlikely that a completely self-adaptive solution would be able to obtain satisfying results. Hence, the user has to be enabled to customize the prompter interface and to change the underlying strategies. This in turn

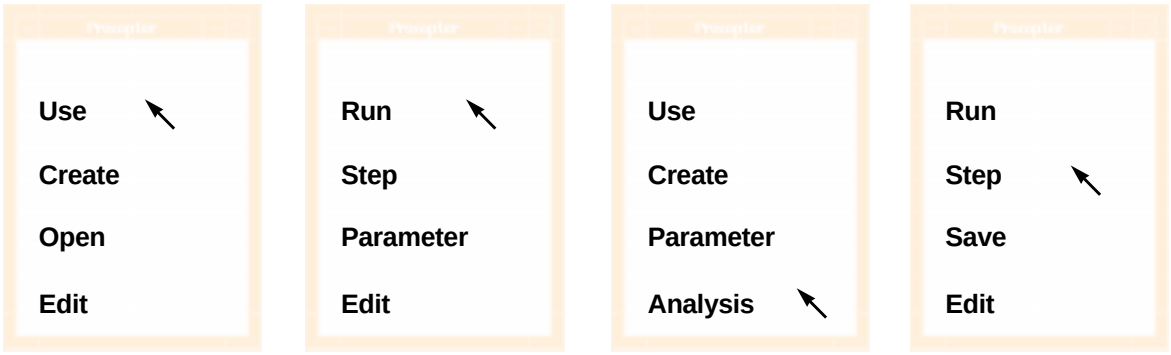

Figure 2. A Sample Interaction Sequence with the Action Prompter

requires a broader interface which provides corresponding access.

The prompter window contains a menu bar and a number of action prompts (figure 1). The menu bar offers three submenus: Context, View, and Options. These menus provide opportunities to customize the prompter interface and to adjust the prompting strategies.

The default number of presented action prompts is 4, due to experiments that determined the average number of items perceivable at a glance to be 4.2 [15]. However, the number of prompts and the layout (rows/columns) can easily be changed by means of a dialog box that is accessible via the view menu (figure 3).

The appearance of the prompts is adjustable. It can be simply the same as in the menu where an action originally appears (figure 2). But also, actions can be extended by the objects to which the actions refer or by task-oriented explanations with respect to the purpose of an action (figure 1).

Figure 3. Prompter Interface Customization

Prompts are exchanged according to context changes. The user can determine when these updates shall occur: “on any context change”, “on any interaction”, or “on action selections”.

Through the action prompter, the user has a good survey of sensible alternatives because the actions in question are listed one beneath another, as opposed to being distributed over different menus. While this means giving up the spacial consistency of static menus, the prompter interface keeps the order of prompts consistent with the original interface and between subsequent updates. If more than one prompt is taken from a menu in the original interface these prompts are presented in the same order as in that menu. Prompts which keep being presented from one update to another remain in their place. If possible without violating these rules, new prompts are presented in the same place as they were presented the last time.

Functional Characteristics

Figure 2 shows a sample interaction sequence using the action prompter to operate a simulation system. After selecting the *Use* action to determine the simulation input data, the actions *Run* and *Step* become available. The user chooses to run the simulation causing a prompt to come up for the *Analysis* action. After selecting this action and carefully looking at the provided analysis of the simulation run, the user decides to step through the simulation by clicking on the *Step* prompt.

The interaction is not as restricted as it might appear at first sight. Even though the user has only four choices at a time, many more action sequences would have been possible. Consider that each action selection results in a potentially different set of prompts. Thus, the shown interaction sequence in this 4-choice, 4-step example is one out of up to 256 possible sequences. In practice, this number is smaller because prompt combinations overlap. But even then there is still more flexibility than required in many situations.

Often, users actually consider only a few actions out of dozens or even hundreds of actions provided by an application. The functional design of the prompter is based on the observation that several reasons can contribute to this fact. Accordingly, the prompter evaluates a number of knowledge sources in order to find appropriate actions to prompt for. Prompting strategies are responsible for extracting the actual prompts as a fusion of the diverse pieces of knowledge. Optional user involvement allows for co-operatively controlling the whole process.

Prompts can appear in virtually any possible combination. There are no pre-canned prompt combinations. However, the prompting strategies are deterministic in the sense that users can rely on seeing the same prompts in the same situation.

The subsequent list of considered knowledge sources and their elements is open, i.e. future work is demanded to add to this list. The prompter design systematically allows for future extensions. In particular, the prompting strategies support an open number of knowledge sources.

Application Model. Sensible alternatives of actions to proceed with are generally determined by the application domain and the tasks users perform in this domain by interacting with the computer. This work uses the UIDE application model [18] that provides information about the elements of the dialog such as objects, actions, relationships between objects and actions, and relationships between subsequent actions. By being used for controlling and monitoring the interaction, this model is able to also supply information about the actual dialog with respect to these elements. If there is a concurrently running application process, such as in supervisory control systems, the model processes and makes available events which are relevant for the interaction and as such important for prompting.

Context Model. In a given situation, relationships between subsequent actions impose restrictions on the availability of particular actions; rule out actions which to choose would not be sensible although they are available; or provide evidence of what actions are to be selected consequently. The user's focus on selected or otherwise used application objects leads to suitable actions when considered in conjunction with the relationships between object and actions. Sets of actions which are often used together in order to perform a certain task or a group of related tasks provide information about a possible next action, given that the majority of recent selections match elements of such a set.

User Model. The users' knowledge and preferences also decide about selections, of course. Novice users know only a subset of all the existing actions of an application. Prompts might be desired only for the known actions to avoid confusion or, in contrast, for unknown actions in order to broaden the user's knowledge. Experts, on the other hand, might have developed preferences for certain actions as far as there are redundant sets of actions and different ways to perform a given task. For purposes of user modeling, this work uses UIDE's built-in mechanisms for recording a statistical history of interactions [16]. In order to obtain a model of the user's expertise, not only successful completions of actions are recorded but also, for instance, related help requests. Preferred actions are determined by a comparative analysis of the user's action-related expertise and the most-recently-used time stamp for that action. The user's overall experience is inferred from how many and how well actions are known and which are preferred.

In the next two sections, the context model and the prompting strategies are described in greater detail. The employed elements of the UIDE model are explained along with this description as far as it is necessary to understand how they are used. For further details on UIDE, the reader is referred to the corresponding papers cited above.

CONTEXT MODEL

The context model is based on the assignment of actions to sets of actions. These sets are an input to the prompting strategies described in a further section. For instance, a prompting strategy might use as an input the set of currently enabled actions and the set of actions which would involve the currently selected object.

These sets can either be predefined or dynamically constructed according to the current state of the interaction. The set of actions which can involve a particular object is predefined as part of the application model. Through the selection of this object, it becomes the set of actions which would involve the selected object. The set of enabled actions, on the other hand, is continuously changing and is obtained by an evaluation of the application model in a given situation. However, only those sets are generated that are actually used by the prompting strategies defined for that application.

The overall context of the interaction at any point in time is given by the entirety of all defined action sets and their composing elements at that time. Three groups of actions sets, i.e. partial contexts, are particularly instructive for adaptive prompting: the *action context*, the *focus context*, and the *task context*.

Action Context

The action context, as defined here, is implicitly given by pre- and postconditions of actions. Pre- and postconditions are part of the application model. They describe under which circumstances an action is executable and what the expected results are with respect to the context of interaction. An action is enabled (i.e., can be executed) if its precondition is fulfilled. The postcondition, on the other hand, determines the context changes which occur when this action has been completed successfully. The application model supports the reasoning about pre- and postconditions, thus allowing for an exploration of relationships between subsequent actions.

Two different facets of the action context are considered and characterized below in terms of which actions go into the corresponding action sets and why they are useful with respect to adaptive prompting. It is also described how these sets are obtained by evaluating pre- and postconditions.

Enabled Actions. Obviously, only actions which are currently enabled are interesting at all for adaptive prompting.

Enabled Actions

actions = set of all actions

disabled = set of all disabled actions

enabled = set of all enabled actions

$enabled_i$ = set of actions which were enabled by one of the *i* most recently executed actions

$actions = enabled \cup disabled$

$enabled_1 \subseteq ... \subseteq enabled_i \subseteq ... \subseteq enabled$

Partly Enabled Actions

pEnabled = set of all partly enabled actions

$pEnabled_i$ = set of actions which were partly enabled by one of the *i* most recently executed actions

$pEnabled_1 \subseteq ... \subseteq pEnabled_i \subseteq ... \subseteq pEnabled$

Enabling Actions

enabling(pEnabled) = set of enabling actions of all partly enabled actions

$enabling(pEnabled_i)$ = set of enabling actions of partly enabled actions $pEnabled_i$

enabling(anAction) = set of enabling actions of the action *anAction*

enabling(aSet) = set of enabling actions of all actions in the set *aSet*

Multi Step Enabling Actions

$enabling(pEnabled_k)$ = set of all enabling actions of partly enabled actions $pEnabled_k$

$enabling_i(pEnabled_k)$ = set of *i*-step enabling actions of partly-enabled actions $pEnabled_k$

$enabling_1(pEnabled_k) \subseteq ... \subseteq enabling_i(pEnabled_k) \subseteq ... \subseteq enabling(pEnabled_k)$

Figure 4. Action Context

Enabled actions are a subset of all existing actions, with the set of disabled actions as the complement.

More interesting are those actions which were enabled by one of the recently executed actions. These actions are sensible candidates to be prompted because the user might have executed the respective action intentionally to enable one of these actions in order to proceed with it. Often, it may be assumed that an enabled action is the more important for prompting the more recently it was enabled. Hence, we define corresponding subsets of all enabled actions. For instance, the smallest subset is the set of actions which were enabled by the most recently executed action. This set is included as a subset in the set of actions which were enabled by one of the two most recently executed actions, and so forth. Figure 4 lists the definitions of the described sets.

Enabling Actions. A specific group among the enabled actions should particularly be considered as prompting candidates, namely all those actions which, if executed, would enable another action. However, it cannot be presumed that the user intends to enable an action only because it is just disabled. More evidence is necessary.

A reasonable assumption seems to be that users may be trying to enable an action which already has been partly enabled by a recently executed action. Partly-enabled means that a precondition has become partly true, for instance, the predicate *A* in the expression *A and B*. Again, it could be considered that the more recently an action was partly enabled the more probably the user wants to proceed with fully enabling this action. Corresponding action sets contain actions which would enable actions which were partly enabled: by the most recently executed action; by one of the two most recently executed actions; etc. See figure 4 for definitions.

A further possibility is to consider not only enabling actions that enable another action in the very next step but also actions that contribute to enabling another action in one of the next steps. As we know from the generation of context-sensitive help on how an action could be enabled [17], there are in general different ways to enable an action, i.e. to make the precondition of an action become true. Planning algorithms can be used to determine possible solutions by looking at which actions' postconditions would satisfy other actions' preconditions. Such planning leads to a number of action paths along which a particular action could be enabled. Some of them might take only one step (as discussed before), others two, three, or even more. The starting action of each path is an enabling action in the broader sense considered in this paragraph.

Correspondingly, the sets of enabling actions discussed above can be orthogonally subdivided into sets which contain starting actions of enabling paths with a certain number of steps. Figure 4 provides the definition for these multiple step enabling actions.

The shorter an enabling path is the more interesting for prompting its starting action is, because of two reasons. Firstly, shorter ways to achieve the same result, i.e. enabling an action, are generally more attractive in terms of performance and simplicity than the longer alternatives. Secondly, short paths do not have as many side effects as long ones potentially have. A short path can therefore more clearly be interpreted as just "enabling an action" while the meaning of a longer path is questionable, at least when only looking at it on the level of pre- and postconditions as opposed to explicitly represented, more task-oriented action sequences.

Prompting for enabling actions is very similar to providing how-to-enable help. With prompting though, the information is embedded in the normal interaction while help usually requires an additional (meta-)dialog. On the other hand, sometimes the more comprehensive information provided by extra help is definitely desired. In addition, if help on how to enable an action is explicitly requested by the user corresponding prompts for the enabling actions would ideally complement the presented help information. This can be

considered in a prompting strategy if, first, an event "help requested for an action *x*" is passed to the prompter, and, second, the set of "enabling actions of the action *x*" is made available.

So far, we discussed two indications that the user might want to enable a disabled action: a partly-enabled action or an action on which the user asks for help on how to enable it. But there are more reasons why enabling actions of certain disabled actions may be considerable candidates for prompting. Consider, for instance, a disabled action which is determined to be a reasonable prompt unless being disabled. Trying to enable this action would make sense. So, prompting for the corresponding enabling actions could be considered. The corresponding set of "enabling actions of all actions in a set" is available.

The presented definition of the action context is exclusively based on a consideration of pre- and postconditions as given by the UIDE application model. Other approaches are conceivable and can be added to the action context definition. For instance, a stochastic model of action sequencing could contribute through providing a set of actions which are likely to be next in a series of successive actions. However, a drawback of a stochastic model is that its outcome is beyond any rationale that allows for understanding and controlling the resulting adaptive behavior. This is in contrast to the context definitions given here which all come along with an intelligible interpretation.

Focus Context

The focus context considers which application objects are in the user's current focus. Obviously in the focus are, for instance, a selected object, an object that has explicitly received the input focus for keyboard events, or an object which was involved in the last selected action. These objects are elements of the *first order focus context*. According information about objects are obtained from the application model.

While the above criteria for focus detection are considered on a syntactical level a more subtle mechanism based on the application model infers further assumptions about the user's focus. This mechanism uses the first order focus context to establish and switch between respective *second order focus contexts* (cf. the notion of the *focus space* in [13]). These contexts include in addition those objects which are closely related to an obviously focused object. The interpretation of "closely related" depends on structural relationships between objects provided by the application model. For instance, in a hierarchical structure all the objects on the path from the root of the hierarchy to the selected object and all the objects logically contained in the selected object would be considered to be in the user's focus. A selected chart within a document within an editor would mean that the editor, the document, and the elements of the chart are also in the corresponding focus context while another document or another chart are not.

Input for the prompting strategies is provided by means of three action sets according to the described focus contexts. See figure 5 for details.

$focused_0$ = set of all actions which involve the currently selected object

$focused_1$ = set of all actions which involve the objects in the current first order focus context

$focused_2$ = set of all actions which involve the objects in the current second order focus context

$focused_0 \subseteq focused_1 \subseteq focused_2$

Figure 5. Focus Context

$active$ = set of actions which are elements of any task contexts

$active_i$ = set of actions which are elements of those i task contexts with the highest activation rates

$active_1 \subseteq ... \subseteq active_i \subseteq ... \subseteq active$

Figure 6. Task Context

Task Context

Task contexts are sets of actions which are often used together in order to perform a certain task or group of related tasks. Task contexts are predefined by either the system designer or the user. They are non-exclusive subsets of all available actions. Based on the monitoring provided by the application model, each task context determines how many of a certain number of recently selected actions match its elements. The resulting value is this task context's current *activation rate*. The number of considered selections depends on the task context and is part of its definition. Task contexts which are expected to be often changed carry a smaller number than those in which the user usually remains for a longer period.

Task contexts are available as input to the prompting strategies. They can be accessed from within a prompting strategy by name or activation rate, for instance, the task context with the highest activation rate, or a union of the two task contexts with the highest activation rates. Figure 6 provides the corresponding definition.

On the one hand, this approach is less expressive than task modeling mechanisms dealing with actual sequences of actions. However, the notion of the task context seems to be just appropriate for action prompting in direct manipulation interfaces. The opportunity to switch between tasks and to choose any possible order of actions provided by direct manipulation dialogs is intrinsically taken into account by the task context approach. In addition, presenting a couple of reasonable alternatives to choose from does not necessarily demand for predicting the very next step in the one and only task the user could be assumed to deal with. Dealing with sequencing in terms of the action context described above might be sufficient in many cases.

PROMPTING STRATEGIES

Defining appropriate strategies to combine all the available information is the key problem of adaptive prompting. Since the user's style of interaction determines what "appropriate" means hardwiring a particular strategy would not help. For instance, the action context information might not be relevant for users who extensively switch between tasks. For them, the task context information becomes important. To complicate things further, a user may even have particular prompting needs for different task contexts, such as for text editing and layout design in a desktop publishing system.

Therefore, the prompter works with an extendable collection of user-adaptable strategies. The actually used strategy is determined in a set-up procedure on the basis of context information. Thus, each prompting cycle consists of two steps:

1. Perform set-up procedure.
2. Perform selected prompting strategy.

The set-up procedure also includes the computation of a base set containing those actions which are to be considered as possible prompts by either strategy (*base*). Often this is simply the set of all actions of an application. It could also be reasonable to exclude actions the user does not know at all or actions with well-known shortcuts. Another set constructed in the set-up procedure contains actions which are definitely required to be prompted for, independently from the applied strategy (*definitePrompts*). This feature allows for dealing with exceptions from the normal prompting strategies. For instance, a certain prompt could be required after the completion of a particular action or in consequence of an external event, such as a power failure alert in an assembly line control application. Another example for such an exception could be a "save" action which might need to be prompted for after every hundred action selections. These two sets can be used in the strategies.

In addition, the set-up procedure can specify modifications to the actual context as to be considered by the strategies. For instance, it could override the activation rate of a particular task context if a certain object is selected or a certain action was completed. The corresponding activation rate would be set to either 0 or 1 in order to avoid or to ensure that a task context is considered, respectively.

A prompting strategy is basically an expression to process the action sets provided by the context model and the set-up procedure. The result of the expression is the proposed set of prompts.

Elements of such an expression are actions sets, set operators, and functions.

Action sets. Beside the named action sets described above, constant and action sets can be used. Intermediate results can be assigned to action set variables for subsequent references.

Set operations. The standard set operations such as union, intersection, and difference are available.

bestKnownActions(
maxPrompts,
$(\text{active}_1 \cup \text{focused}_2) \cap \text{enabled}_3$
)

Figure 7. Sample Strategy

Functions. Arguments and results of functions are actions sets. Available functions include standard set functions and functions provided by the application, context, and user model. For instance, the function to access the enabling actions of an action or set of actions has been described above as part of the context model. The user model provides functions such as *bestKnownActions(aNumber,aSet)* which yields a set of those *aNumber* actions out of *aSet* with the highest user expertise indication.

A simplified example for a prompting strategy is provided in figure 7. This strategy considers actions of the task context with the highest activation rate $active_1$ and the broad focus context $focused_2$. Out of these, actions which were enabled at most three context changes ago are determined as candidates for being prompted. Only the best known actions of these are selected as actual prompts. *maxPrompts* is the maximum number of prompts according to the current prompter appearance.

USER INVOLVEMENT

A set of predefined prompting strategies and a set of stereotype user models for start-up use will normally come along with an application. The user model is individualized while using the application. Thus, the prompter works without any user involvement in the prompting mechanisms.

Sometimes, however, it could be desirable for a user to get an insight into the internal mechanisms of adaptive behavior. Consider, for instance, a case in which the machine intelligence is not able to infer the necessary knowledge, at least not with a reasonable amount of design effort and computational power. The user, however, might be able to fill this gap by a simple change to a strategy or by adding a piece of knowledge to a model. Without the opportunity to become involved, any deficiencies would be irrevocable, the mechanisms, in our case adaptive prompting, would be less useful or even useless in certain situations. Therefore, the user is widely supported in inspecting and controlling the adaptive mechanisms. This approach has been called Computer-Aided Adaptation and described in more detail earlier [6].

It is crucial that the user can obtain any insights and adjustments with a minimal amount of additional knowledge and effort. Therefore, the user interface for all kinds of user involvement consists only of simple menus and dialog boxes. There are no files to edit and no comprehensive languages to learn.

The context menu, for instance, allows the user to explicitly activate a task context (i.e. set activation rate to 1), such as in situations where the prompter failed inferring it correctly

Figure 8. Task Context Editor

Figure 9. User Model Editor

from the dialog or the user unexpectedly wants to switch contexts. Task context definitions can be inspected and easily changed by means of a dialog box (figure 8).

Access to the user model is provided by another dialog box (figure 9). The user can override the system-inferred values which indicate the user's knowledge and preferences with respect to particular actions.

A structure-based editor provides for easy access to the prompting strategies.

EVALUATION

The action prompter always observes the interaction and makes suggestions, whether or not the user actually selects actions from the prompter. As a consequence, an evaluation of the prompting strategies can be achieved by monitoring how often the proposed prompts match the actual user input. With some restrictions, such an evaluation can even be carried out off-line by using session protocols.

This type of evaluation is integrated in the prompter and is performed automatically. The corresponding statistics is accessible through the prompter's options menu.

Further evaluation includes extensive user-testing with respect to questions such as: Do users accept (i.e. use) prompting? Does its usage improve performance and quality of dialogs? How does the optional user involvement affect adaptive prompting?

LIMITATIONS AND RISKS

Adaptive action prompting attempts to provide guidance by presenting and thereby suggesting actions which are assumed to be appropriate in a given situation. However, the provision of guidance has to be designed very carefully since wrong assumptions about the appropriateness of items can cause problems. Obviously, misleading the user would be even worse than no guidance at all. User-testing will have to show whether this problem can be diminished by well-defined, user-controlled prompting strategies.

Using the prompter for subsequent selections in situations with a clear focus is expected to be at least as efficient as using the regular menu. This requires successfully predicting the set of candidates for the next selection with a rate near 100%. The performance could be decreased, however, if the user is too often forced within an interaction sequence to turn away from the prompter to the regular interface because the desired prompt was not offered.

Even if the right prompt is offered the prompter might not always be the best alternative from which to select. For instance, if the user knows exactly where to find an action within a pull-down menu and if, besides, the current pointer position is close to this menu the search for the action in the prompter obviously requires more effort than the selection from the menu. However, his does not argue against the prompter approach as long as users do have the opportunity to select the items from the regular environment.

ADAPTIVE PROMPTING IN MULTIMODAL INTERFACES

Multimodal interfaces aim to relieve users from the burden of dealing with system functions and telling the system "how to do" something. Instead, users will be enabled to communicate their intentions and desires (i.e., "what to be done") to the system in a more natural way by means of gestures and spoken or written natural language. On the long run, users will no longer have to select abstract "tools" or "actions". However, adaptive prompting is even more important and applicable in multimodal interfaces.

There are several observations which support this hypothesis. Most of them can be considered under the two aspects of performance and guidance. A third aspect refers to how adaptive prompting can benefit from multimodality because of the provision of additional focus information and better opportunities for adapting the presentation of prompting information. However, the latter aspect is beyond the scope of this paper.

Performance Aspects

Allowing for more natural, intuitive user input requires a much larger effort for interpreting and evaluating this input within the user interface. Since there are still application programs, functions with parameters, and objects with attributes the user interface has to accomplish a mapping from a fuzzy input to definite entities an application deals with. As a matter of fact, the user interface now has the burden of selecting appropriate applications actions, according to the user's interaction with the system.

Internal Prompting. Adaptive prompting mechanisms can greatly support this internal selection process by providing a list of the actions which are most likely to be selected. Especially in gesture and speech recognition mechanisms, this information could be used to focus on the most probable interpretations resulting in a potentially higher recognition rate.

Confirmative Prompting. As long as recognition rates are still poor, particularly too poor for certain safety-critical application areas, prompting the user should be considered also or even just with multimodal interfaces. Adaptive action prompting provides the user with a cue what the user interface considers to be most likely meant by the next gesture or speech input. On the one hand, the user would know that inputs which are not covered by the menu are more likely to be misunderstood by the interface than those in the menu. On the other hand, the user can use any prompted command (and perhaps gesture symbol) for producing a speech (or gesture) input which will be understood almost for certain. Optionally, the user can choose the desired action from the prompter by means of a more accurate input modality.

The usefulness of this approach has been demonstrated earlier by Tennant et al. [19] who provided a menu to facilitate natural language input and recognition.

Guidance Aspects

Although multimodal interfaces facilitate more intuitive dialogs than ever possible in contemporary direct manipulation interfaces they will not completely relieve the interface from providing guidance. This assumption is based on the every day experience that new capabilities are fully exploited as soon as they appear. Interfaces and underlying applications will become more powerful but hardly simpler to use in their entirety.

Intuitive Guidance. As described above, adaptive prompting provides guidance by particularly offering the most appropriate actions to be proceeded with. Since prompting is a regular part of the user's interaction with the system this kind of guidance fits intuitively into the productive application dialog.

Guidance for Pen-Based Systems. Guidance seems also to be necessary in pen-based systems. For instance, it cannot be assumed that users remember all the possible gestures an interface designer might have thought of as "intuitive". An optional guidance on gestures which are understood by the system would be more than helpful. However, there is rather little screen space on most pen-based systems where to provide this guidance. Adaptive prompting would be a solution since it presents only those items which are most important in a given situation. Hence, space consumption could be limited to what is just necessary.

CURRENT STATUS AND FUTURE WORK

A prototype of the action prompter has been developed in C++ using UIDE [18] for application and user modeling. The prompter interface has been built with SX/Tools [9]. We have evaluated a couple of prompting strategies, however, only in the context of a small demo application. Although first results are encouraging they lack in generality because of the very limited domain. Current work therefore concentrates on developing and evaluating prompting strategies for a prototype of a complex traffic management system. Extensive user-testing will be carried out with this prototype.

We see three major directions for future work:

Prompter Interaction. The interaction with the prompter needs to be further explored in the context of multimodal interfaces and for different appearances of the prompter. We also consider further options for user support. In a future version of the prompter, for instance, the user will be able to choose that an animation of an action selection from the original menu is performed when selecting an action from the prompter.

Considered Knowledge. So far, the prompting strategies have been based on knowledge that is maintained in an advanced user interface environment like UIDE. The intention was to use all available knowledge prior to investigating new knowledge sources. However, further work will include task models and elaborated user models. For instance, the user model needs to be enhanced with respect to contextual correlations of user's preferences.

Prompting Strategies. The prompting strategies are based on expressions for processing action sets. Considering user involvement in prompting strategies design, a representation needs to be found that is easier to understand than formal set operations.

ACKNOWLEDGEMENTS

This work was done at the Graphics, Visualization, and Usability Center during my stay as a visiting researcher at Georgia Tech from September 1992 to August 1993. I would like to thank Al Badre, Jim Foley, Piyawadee "Noi" Sukaviriya, and the graduate students in the UIDE team for many invaluable discussions and comments on the ideas described in this paper. Thanks also to my colleagues Hartmut Dieterich, Uwe Malinowski, and Matthias Schneider-Hufschmidt for providing substantial advice on the topic.

REFERENCES

1. L. Balint: *Adaptive Dynamic Menu System.* Poster Abstracts HCI International '89, Boston, September 18-22, 1989.
2. P. R. Cohen: *The Role of Natural Language in a Multimodal Interface*. Proc. UIST'92, Monterey, CA, Nov 15-18, 1992.

3. A. Cypher. *Eager: Programming Repetitive Tasks By Example*. Proc. CHI ’91, pp. 33-39, 1991.

4. H. Dieterich, U. Malinowski, T. Kühme, M. Schneider-Hufschmidt: *State of the Art in Adaptive User Interfaces.* In: [14].

5. W. D. Gray, W. E. Hefley, D. Murray (eds.): *Proceedings of the 1993 ACM International Workshop on Intelligent User Interfaces*. Orlando, FL. ACM Press, New York, 1993.

6. T. Kühme: *A User-Centered Approach to Adaptive User Interfaces*. In [5], pp. 243-245, 1993.

7. T. Kühme, U. Malinowski, J. D. Foley: *Adaptive Prompting.* Technical Report GIT-GVU-93-05, Georgia Institute of Technology, January 1993.

8. T. Kühme, U. Malinowski, J. D. Foley: *Facilitating Interactive Tool Selection by Adaptive Prompting.* Short Paper Abstracts INTERCHI’93, Amsterdam, The Netherlands, April 24-29, 1993.

9. T. Kühme, M. Schneider-Hufschmidt: *SX/Tools - An Open Design Environment for Adaptable Multimedia User Interfaces*. Proc. Eurographics ‘92, Computer Graphics Forum, Vol. 11, No. 3, pp C-93–C-105, 1992.

10. C. Lewis, P. G. Polson: Cognitive Walkthroughs: *A Method for Theory-Based Evaluation of User Interfaces.* CHI’92 Tutorial Notes, Monterey, CA, May 4, 1992.

11. U. Malinowski: *Adjusting Forms to Users’ Behavior.* In [5], pp. 247-249, 1993.

12. Microsoft Word 5.0, Microsoft Corporation, 1991.

13. M. A. Peréz, J. L. Sibert: *Focus in Graphical User Interfaces*. In [5], pp. 255-257, 1993.

14. M. Schneider-Hufschmidt, T. Kühme, U. Malinowski (eds.): *Adaptive User Interfaces - Principles and Practice*. Elsevier, Amsterdam. *In preparation*.

15. G. Sperling: *The Information Available in a Brief Visual Representation.* Psych. Monogr., Vol. 74, No. 11, 1960.

16. P. Sukaviriya, J. Foley: *A Built-in Provision for Collecting Individual Task Usage Information in UIDE: the User Interface Design Environment.* In: [14].

17. P. Sukaviriya, J. J. de Graaff: *Automatic Generation of Context-sensitive “Show&Tell” Help.* Technical Report GIT-GVU-92-18, Georgia Institute of Technology, July 1992.

18. P. Sukaviriya, J. Foley, T. Griffith: *A Second Generation User Interface Design Environment: The Model and The Runtime Architecture.* Proc. INTERCHI’93, Amsterdam, The Netherlands, April 24-29, 1993.

19. H. R. Tennant, K. M. Ross, R. M. Saenz, C. W. Thompson, J. R. Miller: *Menu-based natural language understanding.* Proceedings of the 21st annual meeting of the Association for Computational Linguistics, pp. 151-158, 1983.